AK

448098
Abstraction of remote operating systems

Disclosed is a method for presenting access to services and resources of a remote operating system, from a local integrated development environment (IDE). Typically, application development IDEs allow programmers to work with and manipulate local programming resources, such as local HTML, Java*, C, and C++ source files. This disclosure documents a consistent and extensible means of adding access to resources that are not on the same local workstation as the IDE.

Abstracting the design for remote access, and authoring a harness framework for that abstraction, offers the ability to easily snap-in addition remote access capilities such that all remote access is consistent for the end user (programmer using the IDE) and such that there is re-use among the various snapped-in tools. Consider some of the many examples of disparate tools that require accessing a remote system, each of which can be found in today's programmer IDEs:

- A remote file system explorer, that possibly allows direct editing of remote source files
- A remote command shell, allowing users to enter commands locally that are executed remotely, with local error feedback
- A remote job or task monitor that allows submitting and trackings jobs or processes on a remote system
- A database administrator tool, that allows DBAs to design and work with databases on remote systems
- A remote debugger for debugging applications on a remote system
- A remote problem determination tool that monitors applications as they run, directing filterable messages back to the local tool
- A tool specialized for working with a particular remote resource, such a message queues, user profiles, message files, etc.

This is just a sample of the many unique tools that exist for accessing and working with remote system resources. This disclosure offers an abstraction that each of these can be mapped to. Some things common to two or more tools in any such list are:

- A means for the user to predefine a "connection" which is a unique identifier for the physical remote system. Typically this is an IP address or hostname, or an http domain name. If each tool defines their own connection, the user ends up with many connection definitions, often to the same remote system. One per tool.
- A means of identifying the remote system daemon or service that will handle this particular tool's requests. This is often an IP port number, which the server side code is waiting on, or the name of a particular servlet or cgi-bin program for http requests. This is typically tool-defined, although sometimes the user is allowed to change it to handle port collision problems.
- Some form of tool-specific communication protocol. It might an industry standard such as JDBC or WEBDAV, or a private program-to-program protocol.
- Some form of filter mechanism, if the data returned from the remote system to the local tool is potentially large. For example, for a remote file system explorer, there might be support for the user to filter the file list by folder or file name or file extension. For remote problem determination, there might be filtering by event, message ID or severity. For database tools, there might be filtering by table or column names.
- A means of identification, such as userld and password. This information might be captured and stored with the connection, or prompted for at connect-time. The server side service uses this for remote login, or access control.

This disclosure is a framework that enables all such remote-access tools to share connections to a particular remote host, and present their user interface in consistent manner. It also builds on that to enable a single remote system view from which the user can drill down, from a list of connections, to each of the tools applicable to that connection.

The framework starts with a list of remote **system types**, such as "Windows", "Unix", "Linux", "OS/400" and "OS/390". There would be a predefined list but there would also be a way for tools to register new system types, and way to query the defined system types. Each system type is simply the unique string identifying it, plus an icon for use in user interface views, plus possibly additional attributes.

The framework then adds a common programming definition for a **connection**. This is simply a unique arbitrary identifier for the connection, the hostname of the remote system, the system type of that remote system, and optionally a userId for the connection. Tools could define additional data to be captured and stored with a connection. The framework supplies the user interface for creating a connection, which is simply a wizard prompting for the information. The framework looks after persisting the list of connections, and supplies APIs for tools to query and manipulate the list of connections.

The framework supplies a registration mechanism for tool providers. Each tool implements a pre-defined interface for a "**subsystem factory**", and registers that factory with the framework, typically using xml. The main job of this factory code (say, a class in Java) is to create **subsystem** objects. A subsystem represents a tool-specific conversation with a remote host. Common attributes most subsystems will have is a port number and user ID for overriding the connection's default user Id. However, each subsystem provider decides on the information to be captured per subsystem.

When a user creates a new connection, the framework polls each registered subsystem factory to see if it is applicable to this connection's system type (eg. "Windows"). If so, the factory is asked to create and return a subsystem instance. The framework maintains a list of all subsystems for a given connection. The factory provider may optionally supply additional UI to support creating more subsystem instances per connection. The framework will provide the user interface for the user to change the attributes per subsystem.

Each subsystem factory can also optionally support **filters**. If so desired, the framework supplies the support for these, and the factory indicates its desire for them by returning true from a framework-defined method. A filter is simply a named list of one or more **filter strings**. How the filter strings are interpreted is subsystem-dependent. The framework just manages the persistence and presentation of the filters, with a unique list persisted per subsystem factory.

Eventually, the actual remote communication has to happen. How that happens is up to the subsystem. The actual live connection is encapsulated in a **system** object, which the subsystems are required to be able to instantiate and return. The system object at a minimum supports the actions of connecting to disconnecting. This typically involves connecting to this subsystem's port (say) at the host identified by the parent connection. How this is done is left to each subsystem. Subsystems must also support a getChildren method that returns a list of objects. What this list is, is up to the subsystem. However, each such object type must be accompanied by an adapter class that knows how to display a given instance of the object type in a tree view. That means responding to requests for the label, the icon, the children and parent of that object. Tool providers will supply this adapter class for their children objects, if they have children objects.

Subsystems also must support, or explicitly state they don't support, the following common actions:

- resolving filter strings (for example, a file subsystem resolves a filter string by returning the list of files matching the string pattern),
- executing a remote command (returning an array of objects often representing the messages generated by the command)
- getting and setting the value of a remote property (what a property is is left to the subsystem to interpret)

Most functionality of most tools can be mapped to this small list of actions. By having common actions, generic tooling and UI can be written that is system-independent and tool-independent. However, tool providers can augment with their own unique actions as required.

That is it. With this core framework, a common remote system explorer can be written. It would be a tree-view with the roots of the tree being the list of defined connections. A menu item or tool bar button allows users to create new connections. When a connection is expanded, all subsystem objects for that connection are displayed. These really represent the tools that know how to work with this connection's system type. When a subsystem is expanded, its children are displayed. The appropriate tool-supplied adapter for each child's class type is consulted to help display the child and potentially displays its children if the adapter reports the child

itself has childr
subsystems chil

If the subsystem
automatically dis
the user interfac
When a filter is
objects automati
framework visua
Other than that,

The subsystem l
filters), but the fr

While the comm
its own rich use
exporer via a to

That is the fram
- there are co
- there are su
- subsystems
- the subsyste
- some comn subsystem
- there is buil filter strings and filter stri

The benefits of t
- A common r
- A common r
- A common against that
- A common additional to

* Java is a Trade

Disclosed by
448098

string identifying t

s simply a unique
/pe of that remote
be captured an
on, which is simply
connections, and

ints a pre-defined
lly using xml Th
stem represents a
is a port number
ler decides on the

ctory to see if it is
eate and return a
ction. The factory
s per connection
tem.

pplies the support
efined method. A
re interpreted is
the filters, with a

subsystem. The
red to be able to
to disconnecting
arent connection
nod that returns a
be accompanied
iew. That means
viders will supply

tions:
g the list of files

sages generated

m to interpret)

actions, generic
ol providers can

d be a tree-view.
allows users to
connection are
m type. When a
ld's class
reports the child

itself has children. The tool provider also supplies the popup menu actions for the subsystem and the subsystems children.

If the subsystem factory indicated it supports filters, then when a subsystem is expanded the framework will automatically display all the filters currently defined for this subsystem's factory. The framework also supplies the user interface for the user to launch the "new filter" wizard. The wizard itself is supplied by the subsystem. When a filter is expanded, the framework asks the subsystem to resolve the filter and displays the resulting objects automatically. Again, those objects must have an adapter registered for their class type to help the framework visualize them in the tree and populate their context menu when the user right-clicks on them. Other than that, the framework does not know or care what the objects are.

The subsystem factory is responsible for saving and restoring its subsystems and other information (except for filters),but the framework will trigger the requests to save and restore at the appropriate times.

While the common tree-view explorer is a benefit to the user, it does not preclude the tool from also supplying its own rich user interface above and beyond the remote system explorer, or perhaps launched from the explorer via a tool-supplied popup menu action.

That is the framework. In summary:

- there are connections and a common connection registry
- there are subsystem factories, that supply subsystem objects when a connection is created
- subsystems represent a particular tool that works on a remote system resource
- the subsystem manages displaying its child objects and its context menu actions
- some common actions are supplied for creating connections, and connecting and disconnecting a subsystem
- there is built-in support for filters, which are named collections of filter strings. The interpretation of the filter strings is tool-dependent, but the framework manages the persistence and presentation of the filters and filter strings.

The benefits of this framework are:

- A common registry for all remote system tools
- A common registry for all remote system connections, so tools can share connections
- A common user interface viewer for exploring a remote system and using all the tools capable of acting against that system
- A common loose definition of a remote system tool, enabling business partners to create and register additional tools that snap-in, and tool and feel like existing IDE-supplied tools.

* Java is a Trademark of Sun Microsystems, Inc.

Disclosed by International Business Machines Corporation
448098

448099
Variable intensity X-ray view box

Currently X-Ray view boxes have two intensity settings, and are lit with fluorescent lamps. The intensity of fluorescent lamps can not be adjusted easily.
I propose using a group incandescent lights indirectly lighting a rough flat white surface inside a view box. The rough surface would be separated from the viewing
surface by several layers of clear Plexiglas and frosted acetate. The viewing surface would be white Plexiglas. The layering will create a even intensity across the viewing surface,
which can be adjusted in a linear mode. The control of the intensity could be either electronic or electrical (dimmer switch). In the case of an electronic control, the intensity could
be used in a DC, fashion or sweeping through the intensity range at a cycle rate just above the persistence of the human eye. A portion of the same circuit could also be used to control
could also be used to control ambient room light. By adjusting the ratio view box intensity to ambient light intensity you could optimize the portion of the image of interest.
Radiologist have developed techniques for looking at that information which is recorded on X-ray film , but not readily discernible to the human eye.
This is a skill built up through years of experience. This invention should allow a less experienced Doctor the ability to view additional information on X-Ray film.

Disclosed by International Business Machines Corporation
448099

448100
Braille overlay for touch screen pin pad/keyboard entry

Disclosed is a device for enabling visually impaired individuals the ability to interact with a flat-panel touch screen for PIN code or text entry. Touch screen have no markings for visually impaired users, causing frustration and disenfranchisement. Providing a removable overlay or template for the touch screen with cutout indicator boxes and braille symbols could allow visually impaired users to interact with the touch screen when entering data.
This overlay could be placed on the touch screen when needed.

Some have placed transparent film with braille symbols permanently over the screen, but this will not work with touch screens that require physical contact with the screen and does not allow for changing input screens such as changing from PIN PAD to alphabetic keyboard or some other such format like a signature capture area or survey form.

Disclosed by International Business Machines Corporation
448100